VULNERABILITIES AND EXPLOITS

Defense Against The Dark Arts

Brad Antoniewicz
Foundstone – McAfee Professional Services
Hi, I’m @brad_anton
BLIZZARDPOCALYPSEMEGAGEDDON!?!
WHAT'S THIS ABOUT NOW?

- **About “Hacking!”**
  - Real Lyfe
  - Trends

- **WinDBG**
  - Win-ternals via firehose
  - How to use

- **Exploitz, son!**
  - How the stack works
  - Exploit Stack-Based Vulnerabilitiy
Program Control

if direction is not blank:
  go(direction);

Please provide a direction
Manipulating Software

- Finding “bugs” which alter the behavior of the program
- Taking advantage of a misconfiguration or poor programming practice
Things change
Defense Against the Dark Arts
Defense Against the Dark Arts
New zero-day vulnerability identified in all versions of IE

The flaw, which is being leveraged in "limited, targeted attacks," allows remote code execution, Microsoft warns.

by Steven Musil @stevenmusil
We’ll give you the vuln, you exploit
WinDbg

Program-Freezer :)
File->Attach->2nd IE Process
Hard Disk

Program
push ebp
mov esp, ebp

Memory

Program
push ebp
mov esp, ebp

WinDbg

Registers

WinDbg r

CPU

Command - Pid 1308 - WinDbg:6.2.9200.16384 X86

0:014> r
eax=7ffad000 ebx=00000000 ecx=00000000 edx=77cff125 esi=00000000 edi=00000000
eip=77c940f0 esp=067eff14 ebp=067eff40 iopl=0 nv up ei pl zr na pe nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000
efl=00000246
ntdll!DbgBreakPoint: 77c940f0 cc int 3

0:014>
### Program Memory

**Program Memory**

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Committed</th>
<th>Private</th>
<th>Total WS</th>
<th>Private WS</th>
<th>Shareable WS</th>
<th>Shared WS</th>
<th>Locked WS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>25,180 K</td>
<td>25,180 K</td>
<td>368 K</td>
<td>2,604 K</td>
<td>276 K</td>
<td>2,328 K</td>
<td>2,244 K</td>
<td></td>
</tr>
<tr>
<td>Mapped File</td>
<td>12,708 K</td>
<td>12,708 K</td>
<td>528 K</td>
<td>528 K</td>
<td>528 K</td>
<td>516 K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shareable</td>
<td>17,788 K</td>
<td>3,912 K</td>
<td>2,260 K</td>
<td>2,260 K</td>
<td>2,260 K</td>
<td>2,252 K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heap</td>
<td>1,728 K</td>
<td>400 K</td>
<td>336 K</td>
<td>320 K</td>
<td>316 K</td>
<td>4 K</td>
<td>4 K</td>
<td></td>
</tr>
<tr>
<td>Managed Heap</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stack</td>
<td>256 K</td>
<td>76 K</td>
<td>76 K</td>
<td>12 K</td>
<td>12 K</td>
<td>4 K</td>
<td>4 K</td>
<td></td>
</tr>
<tr>
<td>Private Data</td>
<td>596 K</td>
<td>28 K</td>
<td>28 K</td>
<td>28 K</td>
<td>24 K</td>
<td>4 K</td>
<td>4 K</td>
<td></td>
</tr>
<tr>
<td>Page Table</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Address**

- `!teb (stack)`
- `!lpeb (heap)`
- `!address`
Program Memory

!teb (stack)
!peb (heap)
!address
InheritedAddressSpace: No
ReadImageFileExecOptions: No
BeingDebugged: Yes
ImageBaseAddress: 013a0000
Ldr: 77d37880
Ldr.Initialized: Yes
Ldr.InInitializationOrderModuleList: 002519f8, 002f4a48
Ldr.InLoadOrderModuleList: 00251968, 002f4a38
Ldr.InMemoryOrderModuleList: 00251970, 002f4a40

Base TimeStamp: Module
13a80000 4c79912 Nov 20 01:46:58 2010 C:\Program Files\Internet Explorer\iexplore.exe
77c60000 4c7b96e Nov 20 04:05:02 2010 C:\Windows\SYSTEM32\ntdll.dll
68d50000 4c7b8e9 Nov 20 04:02:49 2010 C:\Windows\System32\msxml3.dll

SubSystemData: 00000000
ProcessHeap: 00250000
ProcessParameters: 00251108
CurrentDirectory: 'C:\Users\Admin\Desktop'
WindowTitle: 'Microsoft Internet Explorer Default'
ImageFile: 'C:\Program Files\Internet Explorer\iexplore.exe'
CommandLine: '"C:\Program Files\Internet Explorer\iexplore.exe" SCODEF:3736 CREDA1
DllPath: 'C:\Program Files\Internet Explorer;C:\Windows\system32;C:\Windows\sys
Environment: 002507f0

ALLUSERSPROFILE=C:\ProgramData
APPDATA=C:\Users\Admin\AppData\Roaming
CommonProgramFiles=C:\Program Files\Common Files
COMPUTERNAME=WIN-EO5V6K48D0
ComSpec=C:\Windows\system32\cmd.exe
FF_NO_HOST_CHECK=NO
HOMEDRIVE=C:
Program Memory

!address esp

Mapping file section regions...
Mapping module regions...
Mapping PEB regions...
Mapping TEB and stack regions...
Mapping heap regions...
Mapping page heap regions...
Mapping other regions...
Mapping stack trace database regions...
Mapping activation context regions...

Usage: Stack
Base Address: 067e0000
End Address: 067f0000
Region Size: 00004000
State: MEM_COMMIT
Protect: PAGE_READWRITE
Type: MEM_PRIVATE
Allocation Base: 066f0000
Allocation Protect: PAGE_READWRITE
More info: 

!teb (stack)
!peb (heap)
Lab 1: Hello Mr. WinDbg

Viewing Memory: dd, da, du
Breakpoints: bp <addr>
Clear all: bc *
Stepping: t, p

Disassembly: View->Disass.
Conversion: .formats
Math: ?1+1
Modules: lm

Extensions:
Process (inc heap): !peb
Thread (inc stack): !teb
What Addr?: !address
FLAW CLASSES AND VULNERABILITIES
(there are lots)

• Configuration
  – Weak Password

• Logic
  – Authorization Issues

• Storage
  – Inadequate Encryption

• Input Validation
  – Memory Corruption
  – Injection
Accessing memory in an invalid way which results in an **undefined behavior**
Accessing memory in an invalid way which results in an **undefined behavior**

What we’re looking to control

Reading/Writing

Usually Stack or Heap

Originally unintended
Common Categories:

- Lifetime Control
- Uninitialized Memory
- Array index calculations
- Buffer length calculations

Just a few from http://cwe.mitre.org/
Taking advantage of a vulnerability

(Control the “undefined” behavior)
Vulnerability Trigger

Payload

Invokes the software bug to obtain control of the program

Action to be performed when control is obtained
“Shell” code – usually assembly code to execute a shell (e.g. /bin/sh)
Payload
bool isValid = 0;
char buf[255];
memcpy(buf, usrval, len);
...
if (checkPW(buf))
    isValid = 1;
...
return isValid;
Stack Recap

STACK

0xffffffff

0x00000000

Local Variables
Saved EBP
Saved Ret. Addr
Next Func. Param.
Local Variables
Saved EBP
Saved Ret. Addr
Next Func. Param.
main()
1. Step(Left);
2. Step(Right);
3. Step(Left);
4. Step(Right);

Step(FOOT)
A. Lift FOOT foot
B. Move FOOT foot forward
C. Put FOOT foot down
main()

1. Step(Left);
2. Step(Right);
3. Step(Left);
4. Step(Right);

Step(FOOT)

A. Lift FOOT foot
B. Move FOOT foot forward
C. Put FOOT foot down
main()

1. Step(Left);
2. Step(Right);
3. Step(Left);
4. Step(Right);

Step(FOOT)

A. Lift FOOT foot
B. Move FOOT foot forward
C. Put FOOT foot down
main()
1. Step(Left);
2. Step(Right);
3. Step(Left);
4. Step(Right);

Step(FOOT)
A. Lift FOOT foot
B. Move FOOT foot forward
C. Put FOOT foot down
main()

1. Step(Left);
2. Step(Right);
3. Step(Left);
4. Step(Right);

Step(FOOT)

A. Lift FOOT foot
B. Move FOOT foot forward
C. Put FOOT foot down

push ebp
mov ebp, esp
sub esp, 4h
main()

1. Step(Left);
2. Step(Left);
3. Step(Left);
4. Step(Right);

Step(FOOT)
A. Lift FOOT foot
B. Move FOOT foot forward
C. Put FOOT foot down

Local Variables
N/A
Saved EBP
0x000000ff
Func params/Ret Addr
0xffffffff

Main()
main()
1. Step(Left);
2. Step(Right);
3. Step(Left);
4. Step(Right);

Step(FOOT)
A. Lift FOOT foot
B. Move FOOT foot forward
C. Put FOOT foot down

push 0x01001000
call Step()
main()

1. Step(Left);
2. Step(Right);
3. Step(Left);
4. Step(Right);

Step(FOOT)
A. Lift FOOT foot
B. Move FOOT foot forward
C. Put FOOT foot down

push 0x01001000
call Step()
main()
1. Step(Left);
2. Step(Right);
3. Step(Left);
4. Step(Right);

push 0x01001000

Step(FOOT)
A. Lift FOOT foot
B. Move FOOT foot forward
C. Put FOOT foot down
main()
1. Step(Left);
2. Step(Right);
3. Step(Left);
4. Step(Right);

Step(FOOT)
A. Lift FOOT foot
B. Move FOOT foot forward
C. Put FOOT foot down
main()

1. Step(Left);
2. Step(Right);
3. Step(Left);
4. Step(Right);

Step(FOOT)

A. Lift FOOT foot
B. Move FOOT foot forward
C. Put FOOT foot down

push ebp
mov ebp, esp
main()
1. Step(Left);
2. Step(Right);
3. Step(Left);
4. Step(Right);

Step(FOOT)
A. Lift FOOT foot
B. Move FOOT foot forward
C. Put FOOT foot down
main()
1. Step(Left);
2. Step(Right);
3. Step(Left);
4. Step(Right);

Step(FOOT)
A. Lift FOOT foot
B. Move FOOT foot forward
C. Put FOOT foot down
main()
1. Step(Left);
2. Step(Right);
3. Step(Left);
4. Step(Right);

Step(FOOT)
A. Lift FOOT foot
B. Move FOOT foot forward
   Put FOOT foot down

Saved EBP
Saved Ret. Addr
Next Func. Param.
Local Variables
Func params/Ret Addr

EIP
ESP
EBP
Address 2
0x01001000
0x000000fb
0x0ff
0xffffffff
0x0ff
0x00000000
0x000000ff
0x0fb
main()
1. Step(Left);
2. Step(Right);
3. Step(Left);
4. Step(Right);

Step(FOOT)
A. Lift FOOT foot
B. Move FOOT foot forward
C. Put FOOT foot down

defense against the dark arts
																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																												
defense against the dark arts
main()

1. Step(Left);
2. Step(Right);
3. Step(Left);
4. Step(Right);

Step(FOOT)

A. Lift FOOT foot
B. Move FOOT foot forward
C. Put FOOT foot down
main()
1. Step(Left);
2. Step(Right);
3. Step(Left);
4. Step(Right);

Step(FOOT)
A. Lift FOOT foot
B. Move FOOT foot forward
C. Put FOOT foot down

pop ebp
ret 4
main()

1. Step(Left);
2. Step(Left);
3. Step(Left);
4. Step(Right);

Step(FOOT)

A. Lift FOOT foot
B. Move FOOT foot forward
C. Put FOOT foot down
Viewing Call Stack:

k

Baby IDA:
View->Disassembly
Exploit Round 1: Stack Overflow!
func(char *usrval, int len) {
    char buf[4];
    memcpy(buf, usrval, len);
}

char buf[4]
Saved EBP
Saved Ret. Addr
Next Func. Param.
Local Variables
Saved EBP
Func params/Ret Addr
Step(FOOT)
func(char *usrval, int len) {
    char buf[4];
    memcpy(buf, usrval, len);
}

memcpy()...
func(char *usrval, int len) {
    char buf[4];
    memcpy(buf, usrval, len);
}

memcpy()...
func(char *usrval, int len) {
    char buf[4];
    memcpy(buf, usrval, len);
}

c
char buf[4]
func(char *usrval, int len) {
    char buf[4];
    memcpy(buf, usrval, len);
}

EIP → memcpy()…
func(char *usrval, int len) {
    char buf[4];
    memcpy(buf, usrval, len);
}

EIP  memcpy()...
func(char *usrval, int len) {
    char buf[4];
    memcpy(buf, usrval, len);
}

EIP -> memcpy()...
```c
func(char *usrval, int len) {
    char buf[4];
    memcpy(buf, usrval, len);
}
```
func(char *usrval, int len) {
    char buf[4];
    memcpy(buf, usrval, len);
}

EIP  memcpy()...

Defense Against the Dark Arts
func(char *usrval, int len) {
    char buf[4];
    memcpy(buf, usrval, len);
}

memcpy()...
func(char *usrval, int len) {
    char buf[4];
    memcpy(buf, usrval, len);
}
func(char *usrval, int len) {
    char buf[4];
    memcpy(buf, usrval, len);
}
func(char *usrval, int len) {
    char buf[4];
    memcpy(buf, usrval, len);
}
func(char *usrval, int len) {
    char buf[4];
    memcpy(buf, usrval, len);
}

pop ebp
ret 4
func(char *usrval, int len) {
    char buf[4];
    memcpy(buf, usrval, len);
}

pop ebp
ret 4
func(char *usrval, int len) {
    char buf[4];
    memcpy(buf, usrval, len);
}

pop ebp
ret 4

41414141:?????
w000t!

we got program control!
But how do we get code execution?
1. Crash Triage
2. Determine the return address offset
3. Position our shellcode
4. Find the address of our shellcode
• What do we control?
  – What Registers contain attacker-controlled data?
  – What Registers point to attacker-controlled data?
  – Is attacker controlled data on the Stack or the Heap?
  – Do we control critical data such as stack frames?

• Where are we in the execution of the program?
  – Where is the vulnerability?
  – Was the crash caused by an exploit mitigation?
1. Crash Triage
2. Determine the return address offset
3. Position our shellcode
4. Find the address of our shellcode
How many bytes?
• Figure out the offset to EIP overwrite

• Don’t fear javascript :)

• Lab helpers:
  – Built in ‘msfPatternString’ variable
  – From WinDBG:
    • !load byakugan, !pattern_offset 2000
1. Crash Triage
2. Determine the return address offset
3. Position our shellcode
4. Find the address of our shellcode
Time Warp to 1996
LINEAR STACK OVERFLOW

*NOP Sled | Shellcode | Return Address

Vulnerability Trigger/Payload

**usrval**

0x00000000 0xffffffff

Saved EBP

Main()

Local Variables

N/A

Saved Ret. Addr

Next Func. Param.

0x000000ff

Func. params/Ret Addr

41414141

char buf[4]

0x00000000

41414141

Saved EBP

0x000000ff

41414141

Saved Ret. Addr

0x01001000

41414141

Next Func. Param.

0x000000ff

41414141

Func. params/Ret Addr

41414141

41414141

41414141

41414141

41414141

41414141
LINEAR STACK OVERFLOW

Defense Against the Dark Arts
Windows (and modern OS’s)
1. Crash Triage
2. Determine the return address offset
3. Position our shellcode
4. Find the address of our shellcode
DEFENSE AGAINST THE DARK ARTS

LINEAR STACK OVERFLOW

Execution

NOP sled

shellcode

adr guess

Saved EBP

Saved Ret. Addr

char buf[4]

Next Func. Param.

Local Variables

Func pointer Ret Addr

Execution Good for *nix (consistent stack)
What memory address is this at?
func(char *usrval, int len) {
    char buf[4];
    memcpy(buf, usrval, len);
}

pop ebp
ret 4
func(char *usrval, int len) {
    char buf[4];
    memcpy(buf, usrval, len);
}

pop ebp
ret 4

41414141:?????
Trampoline!

Jump to a location which jumps to another (e.g. addr of a jmp esp instruction)

AND PHYLLIS WAS NEVER INVITED TO ANOTHER TRAMPOLINE PARTY AGAIN
1. Find a module loaded at a static address
2. Find “jmp esp” (or similar instruction) within that memory space
0:012> u 77c373cb
ntdll!RtlpConsole+0xd4:
77c373cb 2a81ffe40400    sub al,byte ptr [ecx+4E4FFh]
77c373d1 00742281        add byte ptr [edx-7Fh],dh
0:012> u 77c373cb
ntdll!RtlpConsole+0xd4:
77c373cb 2a81ffe40400   sub    al,byte ptr [ecx+4E4FFh]
77c373d1 00742281   add    byte ptr [edx-7Fh],dh
0:012> u 77c373cb
ntdll!RtlpConsole+0xda:
77c373cb 2a81ffe40400   sub   al,byte ptr [ecx+4E4FFh]
77c373d1 00742281        add   byte ptr [edx-7Fh],dh

0:012> u 77c373cd
ntdll!RtlpConsole+0xda:
77c373cd ffe4            jmp   esp
77c373cf 0400            add   al,0
0:012> u 77c373cb
ntdll!RtlpConsole+0xd4:
77c373cb 2a81ffe40400 sub al,byte ptr [ecx+4E4FFh]
77c373d1 00742281 add byte ptr [edx-7Fh],dh

0:012> u 77c373cd
ntdll!RtlpConsole+0xda:
77c373cd ffe4 jmp esp
77c373cf 0400 add al,0
func(char *usrval, int len) {
    char buf[4];
    memcpy(buf, usrval, len);
}

pop ebp
ret 4
func(char *usrval, int len) {
    char buf[4];
    memcpy(buf, usrval, len);
}

pop ebp
ret 4

Ntdll.dll (0x77c373cd)

EIP  jmp esp
func(char *usrval, int len) {
    char buf[4];
    memcpy(buf, usrval, len);
}

pop ebp
ret 4
func (char *usrval, int len) {
    char buf[4];
    memcpy(buf, usrval, len);
}

defense against the dark arts

pop ebp
ret 4
func(char *usrval, int len)
char buf[4];
memcpy(buf, usrval, len);
}
pop ebp
ret 4
Ntdll.dll
EIP  jmp esp
shellcode
saved ebp
saved ret
addr
41414141
41414141
77c373cd
41414141
Lab 2: Smashing the Stack!

1. Triage:
   - ‘k’ for call stack and disassembly view
   - ‘bp <addr>’ for break points, ‘t’ to step into

2. Trigger (build the ‘s’ variable in the JS)
   - MakeString(Amount); // 1 = 2 bytes
   - Remember order (12345678 = \u5678\u1234)

3. Find address to jmp esp in windbg, add it to ‘s’
   - s [start] [end] ff e4

4. Add in ‘shellcode’ variable to ‘s’